INTERNATIONAL JOURNAL OF INNOVATIVE RESEARCH & DEVELOPMENT

ISSN 2278 – 0211 (Online)
ISSN 2278 – 7631 (Print)

Task Migration in Cloud Computing

Professor Dipti Ajmire
G.H. Raisoni College Of Engineering, Nagpur, India
Dr. Mohammad Atique
Associate Professor, S.G.B., Amravati University, India

Abstract:
Heterogeneous Process Migration is a technique whereby an active process is moved from one machine to another of possibly different architecture. This necessitates the capture of the process's current state of execution and recovering it on the destination machine in a manner understandable to it. My work mainly focuses on capture and recovery of the internal state of process, comprising of the execution and data state – activation history, and static and heap data. The two major issues involved here are- the mechanism of process state capture and recovery and the initiation of the capture mechanism.

Key words: *Process Migration, Latency*

1. Introduction

Heterogeneous Process Migration is a technique whereby an active process is moved from one machine to another of possibly different architecture. This necessitates the capture of the process's current state of execution and recovering it on the destination machine in a manner understandable to it. My work mainly focuses on capture and recovery of the internal state of process, comprising of the execution and data state – activation history, and static and heap data. The two major issues involved here are- the mechanism of process state capture and recovery and the initiation of the capture mechanism.

A major issue of process state capture in heterogeneous distributed computing systems is that it cannot simplify be initiated instantaneously, once a request for capture has been received. The capture can be initiated only at certain points – the points of equivalence between different instances of computation of different architectures – so that the process can be restarted at exactly at the same point where it was paused. Ideally, once the request for capture has been received, the state capture should be initiated at the very nest point of equivalence encountered. At the same time, it should be ensured that the performance overhead incurred during normal execution should be insignificant. We propose a novel approach to process state capture and recovery, which achieve the above objectives.

In the case of high performance computing applications, the performance gain achieved by the elimination of polling, especially within critical loops, would be significant. Also, our solution to the heterogeneous process state capture problem is capable of effectively enabling all potential points of equivalence present in a computation if minimal latency is desired. In a polling approach, to achieve this minimal latency, poll- points would have to be placed at all potential points of equivalence, in which case, the performance overhead incurred during normal execution would reach severely unacceptable levels.

One of the central features of our approach is automatic transformation of program code to incorporated state capture and recovery functionality. This program modification is performed at the platform-independent intermediate level of code representation, and preserves the original program semantics. The attractive properties of this approach include portability, ease of use and flexibility with respect to basic performance trade-offs and application-specific requirements

2. Heterogeneous Process Migration

Process Migration can be defined as the ability to move a currently executing process between different processors which are connected only by a network (that is, not using locally shared memory). The Process Migration mechanism must package the entire state of the process executing in the originating machine so that the destination machine may continue its execution. The process should not normally be connected by any changes in its environment other that in obtaining better performance.

Research in to the field of Process Migration has concentrated on efficient exchange of the state information. For example, moving the memory pages of process from source machine to the destination, correctly capturing and restoring the state of process(such as

registered contents)., and ensuring that the communication links to and from the process maintained.
Careful design of an operating system's IPC mechanism can be ease the migration of process. Most Process Migration systems make the assumption that the source and destination host have the same architecture. That is, their CPU understands the instruction set, and their operating systems have same set of system calls and the same memory conventions. This allows state information to be copied verbatim between the hosts, so that no changes need to be made to the memory image. Heterogeneous Process Migration remove this assumptions, allowing the source and destination hosts to differ in architectures. In addition to the homogeneous migration issues, the mechanism must translate the entire state of process so it may be understood by the destination machine. This required knowledge of the type and location of all data values (in global variable, stack frame and on the heap).[2]

2.1 Applications
The traditional reasons for using Process Migration have been identified as:

2.1.1. Load Sharing Among A Pool Of Processors
For a process to obtain as much CPU time as possible, it must be executed on the processor that will provide the most instructions and I/O operations in the smallest amount of time. Often, this will mean that the fastest processors as well as those executing a small number of jobs will be the most attractive. Migration allows a process to take advantage of underutilized resources in the system, by moving it to a suitable machine.

2.1.2. Improving Communication Performance
If a process requires frequent communication with other processes, the cost of this communication can be reduced by bringing the processes closer together. This is done by moving one of the communicating partners to the same processing nodes the other (or perhaps to nearby processing node).

2.1.3. Availability
As machines in the network become unavailable, users would like their jobs to continue functioning correctly. Processes should be move away from machines that are expected to be removed from service. In most situations, the loss of the process is simply an annoyance, but at other times it can be disastrous (such as an air traffic control system). Reconfiguration – while administering a network of computers, it is often necessary to move services from one place to another (for example, a name server). It is undesirable to halt the system for a large amount of time in order to move a service. A transparent migration system will make changes of this kind unnoticeable.

2.1.4. Utilizing Special Capabilities
If a process will benefit from the special capabilities of a particular machine, it should be executed on that machine. For example, a mathematics program could benefit from the use of special math coprocessor, or an array of processors in a supercomputer. Without some type of migration system, the user will required to make their own decision of where to execute the process. Often users will not even be aware of their program's special needs.

2.1.5. Mobile Computing
Mobile Computing is a term used to describe the use of small personal computers that can easily be carried by a person, for example, a laptop or handheld computer. To make full use of these systems, the user needs to be communicating with larger machines without being physically connected to them, normally done via wireless LANs or cellular telephones. It has been proposed that process migration is important in this area. For example, users may active a program on their laptop, but in order to save battery power or to speed up processing may later choose to transfer the running process onto a larger compute server. The process would be returned to the smaller machine to display result. These concepts can be extended to allow a program to move between workstations as its owner moves. A person may be using home computer, with large number of windows on their screen. By remotely connecting to the computer at their place of work, they will be able to continue executing those programs in their office. If they choose to move between offices, the window system (and programs) could potentially follow them.

2.1.6. Wide Area Computing
For a computer to be part of the internet, it must understand the internet communication protocols. Since there are no constraints on other software, such as operating systems and programming languages, an enormous amount of heterogeneity exists. The one limitation of global computing which will never be resolved is the propagation delay that is suffered over wide area networks. At best, data can only be transmitted at the speed of light, causing noticeable delays. If a program makes frequent use of remote data, its performance will suffer. Process migration can help alleviate this problem by moving the program close to the data, rather than moving the data to the program.
Typically, the program would start executing on the user's local machine. If it later makes frequent accesses to remote data, the migration system will reduce the delay be moving the process to a machine that is physically closer to the data. This makes the most sense in the case where the program is smaller than the data.

2.2. The Heterogeneous Process State Capture Problem

A mechanism solving the heterogeneous process state capture and recovery problem must provide the ability to generate a checkpoint for an active process- a complete description of that process's state and point in execution. The mechanism must also support the later use of that check point to restart a process with equivalent point in execution, possibly on different type of computer from the one on which the original checkpoint was created. Figure 1 depicts the basic operation of a heterogeneous state capture and recovery mechanism.

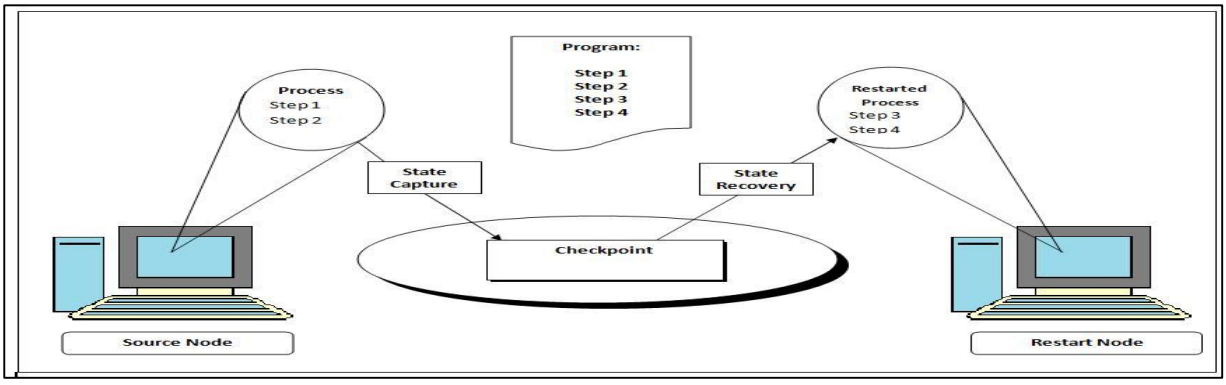

Figure 1: Operation of Heterogeneous State Capture and Recovery Mechanism

The possibility of cross-platform restart leads to the most fundamental solution constraint: the mechanism should generate platform independent checkpoints – i.e. checkpoints produced on computer system of any architecture and operating system should be recoverable on a system of any other architecture and operating system.

3. Task Migration

A major issue of process state captures in heterogeneous computing systems is capture initiation. Current approaches incur significant performance overhead during normal execution of the process (i.e. when state capture/recovery is not being performed) in order to ensure proper initiation of state capture. This is because of their introduction of instructions into the user code, either to poll for a capture request, or to ensure correctness of self modifying code in the case of a poll-free mechanism. In this paper, we propose a fundamentally new approach to heterogeneous process state capture and recovery that achieve minimum performance overhead during normal execution by obviating the introduction of such instructions.

In the case of high- performance computing applications, the performance gain thus achieved- especially within critical loops-would be significant. Also, our solution is suitable for effectively enabling all potential points of equivalence present in a computation if minimal latency is desired. [2]

3.1. The Issue of Heterogeneous Process State Capture

A mechanism solving the heterogeneous process state capture and recovery problem must provide the ability to generate a checkpoint for *active process* – a complete description of that process's state and point in execution- and also support the late use of that checkpoint to restart a process with equivalent state and at an equivalent point in execution, possibly on a different platforms from the one on which the original checkpoint was created [4]. A major issue of process state capture in heterogeneous computing systems is its initiation, once the request has been received. The process cannot simply be paused (for capturing its state) at any point of its execution, but can only be paused at points that have *equivalent points* in all the instances of the same computation on different platforms.

There are many possible points of execution equivalence that can be identified in any program. But usually, not all of these candidates would be effectively enforced as actual points of execution equivalence. This is because, in order to ensure consistency, machine dependant optimizations would need to be disabling across such enable pots of equivalence, which result in performance degradation. The selection of the subset of points of equivalence to be enabled is based on a trade-off between the performance overhead incurred during normal execution and the wait-time from request to actual initiation of the capture. Also, the possibility of cross=platform recovery lead to the most fundamental solution constraint: the mechanism should capture the state in platform- independent manner – i.e., checkpoints produced on a computer system od any architecture should be recoverable on a system of any other architecture. For example, one straightforward approach is to use an interpreted language. In these designs, the interpreter acts as a virtual machine that can artificially homogenize a system composed of heterogeneous elements. Unfortunately, such schemes severely compromise performance since they run at least an order of magnitude slower than their native code counterparts. Therefore, in our intended environment, processes run on nodes, typically executing in native code form due to performance considerations. In homogeneous systems, process state capture and recovery mechanisms can simply and directly copy the state of the process verbatim, without semantic analysis and interpretation of that state. Unfortunately, in a heterogeneous environment, the state of process cannot be

captured using this native approach because of difference in instruction sets and data representation. To mask the varying feature of processes in a heterogeneous system, a state capture mechanism must examine and capture the logical internal structure and meaning of the process state – i.e. the logical point in execution, the call stack (or call stacks, if threads are supported), complex data structures, the logical structure, and contents of heap allocated memory, and all other process state must be analyzed and captured in a platform-independent format.

3.2. Related Work

Although, process sate capture/recovery mechanism for homogeneous computing systems are well developed and can now typically be performed with minimum overhead and latency, much less progress has been made towards providing such functionality across heterogeneous architectures. Because of the additional inherent complexity i9ntroduced by heterogeneity, very few designs for such facilities have been developed to date.

In applications such as process migration due to load balancing policies, or logging mechanism for fault tolerant systems, arbitrary long latencies would not be acceptable. In the case of load balancing, for example, the very purpose of migrating the process itself is to reduce the load on the system as soon as possible. For such applications with a minimal latency requirement, almost all potential points of equivalence present in the computation must be effectively enabled. The polling approach would not be suitable because the performance overhead due to persistence polling at all such points would reach severely unacceptable levels.

3.3. Objectives

- Efficiency: As a goal, in addition to providing efficient state capture and recovery, the run-time performance overhead introduced by the mechanism should be acceptable levels. In particular, if checkpoints are not performed during a certain period of execution, a process with state capture and recovery service available to it should not run significantly, slower than the version of the code without this service available over the same period.
- Generality: The mechanism should be appropriate for used with a wide range of architectures and wide variety of programs that are written in variety of languages, and that solve the wide range of problems.
- Suitability for Minimal Latency: The state capture mechanism should be suitable even for ensuring minimal possible latency (the time delay between when a capture is scheduled or requested and when a capture is actually initiated) which is the time taken to reach the very next *possible* point of equivalence in the computation.
- Ease of Use: When possible, the mechanism should be fully automatic, requiring little of no effort on the part of the application programmer. Such full automation should be possible for programs expressed in a platform-independent manner, i.e. programs that do not rely on specific hardware or software features of certain computer systems.

Now to implement the task migration algorithm, the following assumption should be considered first

3.4. Assumptions

- Compiler support is available for obtaining the equivalence point table.
- Operating system support is available for obtaining the current value of the program counter for a process.
- Machine dependent optimizations across enabled points of equivalence shall not be allowed since they must prevent the different complied versions of the program across heterogeneous platforms from hitting every such point consistently.

3.5. State Capture Initiation Algorithm

When a request for capture of a process's state is received from a external agent, the following steps are carried out:

- The current value of the program counter is obtained using operating system support.
- The program counter value is used to identify the currently executing function and the current block (the segment of code between two points of equivalence containing the current instruction).
- The following steps are carried out for ensuring that the state capture is initiated on encountering the next point of equivalence:
    - If the program counter is exactly at a point of equivalence, an instruction to initiate state capture is placed at the same point
    - If no program control instruction lies between the current point of execution and the sequentially next point of equivalence, an instruction to initiate state capture is placed at the sequentially next point of equivalence.
    - Else, we copy and pass control to the code fragment lying between the current point of execution and the sequentially next point of equivalence, with the following modifications made for each program control instruction lying within that fragment:
    - Code is placed in place of the program control instruction to ensure that the point of equivalence that would have been encountered next, if the program control instruction were allowed to execute, is registered.
    - An instruction to initiated state capture is then placed at the end of that code,

Most importantly, the program control instruction is not allowed to actually execute- the steps that it would have carried out if it would have executed (for example, setting up a new activation record in the case of procedure call instructions), will be made to carried out, except for passing control to some point, instead of which the control is passed to the state capture mechanism. The

process is now "informed" to initiate the state capture as soon as possible. Once the process start executing latter, it eventually encounters a point of equivalence. Here, the initiation instruction (which is call to the state capture mechanism) gets executed and the control is thus transferred to the state capture mechanism.
This algorithm requires that all points in the code which are possible destinations to jump instructions should be enabled as points of equivalence. This may not be a restrictive requirement, as optimizations would anyway not be performed across jump destinations to preserve program correctness. [4]

3.6. State Capture Algorithm

3.6.1. Once the Control Is Transferred To the State Capture Mechanism Function, the Following Task Are Performed Initially

- The point of equivalence at which and the interrupted function within which) capture has been initiated is noted.
- The return address of the current activation record is replaced by the address of the saving epilogue of the interrupted function. Finally, a return is performed so that control is passed to that saving epilogue.

3.6.2. The Saving Epilogue Performs the Following Tasks

- Save the local variable and actual parameters present in the current activation record.
- Identify the caller of the current function using the return address available in the current activation record. The point of equivalence preceding the point of execution of the calling function is noted.
- The return address of the current activation record is replaced by the address of the saving epilogue of the caller function. Next, a return is performed so that control is passed to the caller's saving epilogue.

3.6.3. Step B Is Repeated Until All Activations Are Saved

3.6.4. When the bottom of the activation history stack is reached, the epilogue also performs the saving of the static (global) data and the heap data

While saving activation, static or heap data, the state of pointer is captured according to its logical meaning (i.e., the data object or code point to which it is pointing) rather than its value indicating the physical address [8, 9, and 10]. In order to identify the data object being pointed, given physical address, we required one of the following:

- Compiler-support in terms of information about the positions of global within the global data area and the activation record structures of the various functions, as well as run-time support only for registering heap data.
- Run-time support for registering local, global and heap data.

For all other data types, the data is copied verbatim into the checkpoint. [4]

3.7. State Recovery Algorithm

Once a new process has been created on the destination machine, the following steps are carried out to perform the process state recovery:

3.7.1. A Jump Instruction with Destinations as the Corresponding Restoring Epilogue Is Placed At the Entry Point of Each Function Present In the Program

3.7.2. When the First Activation Record (For The Base Function) Is Created, The Corresponding Epilogue Will Restore The Static (Global) And Heap Data From The Check Point File

3.7.3.The Restoring Epilogue Performs The Following Tasks

- Restore the local variables and actual parameters into the current activation record.
- If there are no more activations to be restored: The original entry point of each function is restored back.
- A jump takes place to the point in the destination's object code corresponding to the point of equivalence (at which this function activation's execution was frozen) noted during state capture.

3.7.4. Step C Is Repeated Until All Activations Have Been Restored

This happens automatically since the point to which the jump takes place will contain a call instruction until all the activations record have been restored. (Once all the activations have been restored, the original function entry points are restored and control is transferred to the point of equivalence at which the state capture had been initiated). [4]
The process finally resumes normal execution. Again, while restoring the activation, static or heap data, the state of pointer is mapped back from its logic meaning to its value indicating the physical address on this machine. For all other data types, the source data copied into the checkpoint in now translated accordingly into the destination architecture data format mapping functions, if necessary. This is in accordance with the "receiver makes right" policy. This policy has the advantage that only one translation needs to be done (by the receiver) and there is no need for any intermediate data format representation. Also, if the state is to be recovered on the same architecture as the source, there is no need for any translation at all.

4. Conclusion

This paper presents an approach to process state capture and recovery in heterogeneous computing systems that achieve minimum performance overhead during normal execution of the process. The solution presented, being poll-free, is suitable even for an application desiring minimal latency as it can afford to effectively enable all potential points of equivalence present in a computation. Also high performance computing applications can perform significantly better due to the reduced performance overhead, especially within critical loops.

5. References

1. Prashanth P. Bungale, Swaroop Shridhar. ”An Approach to Heterogeneous Process State Capture/ Recovery to Achive Minimum Performance Overhead during Normal Execution.” Proceeding of the International Parallel and Distributed Processing Symposium(IPDPS’03)
2. Ramon Lawrence, “A Survey of Process Migration Mechanism,” Student Report for course project 1997, University of Manitoba.
3. M.J. Litzkow, M. Livny, and M. W. Mukta,” Condor-A Hunter of Ideal Workstation,” in Proceeding of the Eight International Conference on Distributed Computing Systems, pp. 104-111,1998.
4. Adam John Ferrari, “Process State Capture and Recovery in High Performance Heterogeneous Systems”, Ph.D. Thesis, Department of Computer Science, University of Virginia, January 1998.
5. C.Schroth and T. Janner, (2007). Web 2.0 and SOA: Converging Concepts Enabling the Internet of Services. IEEE IT Professional Vol.9, No.3 (pp.36-41), June 2007.
6. P.A Laplante, Zhang Jia and J.Voas, “What's in a Name? Distinguishing between SaaS and SOA," IT Professional” vol.10, no.3, pp.46-50, May-June 2008.
7. Aaron Weiss (2007), Computing in the Clouds, net Worker, Dec. 2007, 11(4):16-25.
8. C.Schroth and T. Janner, (2007). Web 2.0 and SOA: Converging Concepts Enabling the Internet of Services. IEEE IT Professional Vol.9, No.3 (pp.36-41), June 2007.
9. Qiang Li1,2, Qinfen Hao1, Limin Xiao1, Zhoujun Li1, “Adaptive Management of Virtualized Resources in Cloud Computing Using Feedback Control” ,ICISE2009.
10. Wenzheng Li, Hongyan Shi(2009), College of computer and information Engineering Beijing Technology and Business University Beijing, “Dynamic Load Balancing Algorithm Based on FCFS”,IEEE fourth International Conference on Innovative Computing, Information and Control.
11. Zhenhuan Gong, Prakash Ramaswamy, Xiaohui Gu, Xiaosong Ma(2009), “SigLM: Signature-Driven Load Management for Cloud Computing Infrastructures”IEEE.
12. Greg Boss, Padma Malladi, Dennis Quan, Linda Legregni and Harold Hall, Cloud Computing. http://www.ibm.com/developerworks/websphere/zones/hipods/.
13. Verwoerd T,Hunt R.GLOB:Generic Load Balancing Networks. Proceedings of Ninth IEEE International Conference, 2001
14. Li Wenzheng, Guo Qiao, Wang Li, Guo Weimin, “Study and Realization of Internet Server Load Balancing”, Computer Engineering, 2005, vol. 31, No. 6
15. Cao Z,Wang Z,Zegura E W.Performance of hashing-based schemes for internet load Balancing[C].Proceedings of IEEE Inform,2000 [12] O.Conlan, "The Multi-Model, Metadata Driven Approach to Personalised eLearning Service", Dissertation, University of Dublin, Ireland.
16. Zhou S.A Trace. Driven Simulation Study 0f Dynamic load Balancing [J] . IEEE Trans on Software Engineering,1998,14(9)
17. Rajkumar Buyya, Chee Shin Yeo, and Srikumar Venugopal, Market-Oriented Cloud Computing: Vision, Hype, and Reality for Delivering IT Services as Computing Utilities, Proceedings of the 10th IEEE International Conference on High Performance Computing and Communications, Dalian, China, 2008, pp.5-13.
18. Foster, Z. Yong, I. Raicu, and S. Lu, “Cloud Computing and Grid Computing 360-Degree Compared,” Grid Computing Environments Workshop, Nov, 2008.
19. Repast Organization for Architecture and Development, http://repast.sourceforge.net, 2003 (Accessed 10th June 2009)
20. The Canadian Encyclopedia, Retrieved from http://www.The canadianencyclopedia.com/index.cfm? PgNm= TCE & Params =AISEC819777.
21. E. Di Nitto, D.J. Dubois, R. Mirandola, F. Saffre and R. Tateson, Applying Self-Aggregation to Load Balancing: Experimental Results. In Proceedings of the 3rd international Conference on Bioinspired Models of Network, information and Computing Systems (Bionetics 2008), Article 14, 25 – 28 November, 2008.